

NASA Technical Memorandum 87577

TRICCS: A PROPOSED TELEOPERATOR/ROBOT INTEGRATED COMMAND AND CONTROL SYSTEM FOR SPACE APPLICATIONS

**(NASA-TM-87577) TRICCS: A PROPOSED
TELEOPERATOR/ROBOT INTEGRATED COMMAND AND
CONTROL SYSTEM FOR SPACE APPLICATIONS (NASA)
31 p HC A03/MP A01**

CSCL 09B

N85-35637

**Unclass
22297**

63/61

RALPH W. WILL

JULY 1985



**National Aeronautics and
Space Administration**

**Langley Research Center
Hampton, Virginia 23665**



INTRODUCTION

A wide variety of robot programming languages (refs. 1-10) are available today, but it has been estimated (ref. 7) that in 1982, less than 10 percent of the robots on the factory floor had actually been programmed. Most are manually taught. This situation is attributed to a variety of deficiencies in these languages (ref. 1). These deficiencies include excess complexity, poor communications with external devices, configuration dependency, lack of portability, and no provision for concurrent activities or multiple arms. Some of these difficulties are associated with the languages themselves and some, such as processor and network architecture problems, are related to the robot's real-time control or operating system. In any event, robot programming languages are now apparently more trouble than they are worth.

One thing that makes robot programming languages so cumbersome to use is the fact that most are totally ignorant of the physical environment in which the robot is operating. Their data types are scalars and vectors while the robot is dealing with spatial points, orientations, planes, and objects of various shapes. Several investigators (refs. 7-10) have recognized the limitations of robot programming languages without environment or world models and have considered schemes for capturing and utilizing this information.

Another problem is the fact that most robot programming languages are quite rightly directed toward the automated factory or assembly line applications. Little effort has been given to addressing the somewhat special requirements of space operations. The basic purpose of robots in space is to venture into a hostile environment and, in conjunction with man (at least for the near term), to perform repair, test, and assembly operations. Such tasks are definitely not of a repetitive, assembly line nature which lend themselves to manual teaching techniques. This implies the need for a powerful, high level command language with the ability to describe and deal with an undefined environment. The tasks will also not be fully specified and will involve new and unexpected situations. This requires a very flexible system which is able to adapt to different modes of operation and to operate at several levels. Many space operations, partly due to the lack of gravity, require two robot arms, which implies coordination algorithms, concurrent processing, and sensor feedback.

Programming and checkout in the robot programming language must be simple, fast, and preferably interactive so that programming time does not exceed task performance time. Otherwise, nothing is gained over the teleoperator system. This is particularly true for space operations which may be performed only once rather than repeated as on an assembly line. The system will be used by people who are trying to perform complex, precise jobs via remote control. They are neither machine tool operators nor computer scientists, so the available commands need to be clear, powerful, and most importantly, very task oriented. The command set should also be readily expandable via user-defined procedure and function calls. The ultimate goal should be to incorporate an expert system/planner to further increase man's command power and efficiency. Consideration must be given in the programming language to the fact that the planner must decompose its high level commands into robot programming language commands and task invocations.

The requirement for fast, interactive programming and checkout conflicts with what is known about the difficulty of describing concurrent operations and debugging

concurrent programs. Thus, some clear, unambiguous means of describing arm coordination and simultaneous motions is needed. Also, arm collision avoidance, previously left largely to the user, had better be implicit and rigorous. System error recovery, largely ignored by robot programming languages, becomes a necessity.

The high cost of space operations dictates a reliable system which is capable of getting the job done somehow. This also requires a flexible system with backup modes and alternatives with manual interaction at as many levels as possible. The role of man has not been defined for space related robotics systems, but it will probably represent an evolutionary progression from a "hands-on" teleoperator toward an executive dealing with an expert system as robot systems get more sophisticated. It is important to devise a system now which is capable of expanding and evolving and which has features to address future needs.

The Teleoperator/Robot Integrated Command and Control System (TRICCS) has two basic objectives: (1) to address some of the deficiencies of robot programming languages, particularly in the area of robotic system interfaces such as environment knowledge base and sensor feedback; and (2) to investigate a powerful, flexible, higher order language system capable of being used at several levels and of supporting the evolution of a space related teleoperator/robotic system. This means that the language must contain world model variables, sensor variables, constructs designed for interactive use, and provisions for multiple arm and concurrent operations. TRICCS has been designed to be embedded in a goal-directed robotic system involving expert systems, an environment knowledge base, a vision system, sensor feedback, and a manual interface where the operator uses TV and computer-generated displays.

Figure 1 shows a simplified block diagram for a goal-directed robotic system. This system is composed of several hierarchical levels, each of which, for operational flexibility, has been provided with an operator interface. The lowest level is a teleoperator in which continuous manual motion commands from switches or hand controllers are transformed into individual joint commands to produce the desired robot operation. On the second level, the operator becomes a programmer, communicating with the robot programming language via a keyboard. Here, a primary concern is loading the environment or domain knowledge. Two sublevels are provided: (1) a sophisticated vision system which searches for, identifies, and locates parts and objects based on tabular CAD/CAM type information and directives in the program, and (2) a manual object identification and definition mode if no vision system is available or if no CAD/CAM data exist for particular items. The manual definition mode, described in the Declaration and Definition of Variables Section, is cumbersome and time consuming, but it is treated in some detail here since no implementable vision system exists and a manual mode is essential for flexibility.

The third and highest level of the goal-directed robot involves an expert system/planner which is defined for our purposes to be any program that uses goals supplied by an executive operator to perform complex tasks. These tasks are accomplished with the aid of a fact and rule knowledge base which might also be supplied by CAD/CAM information on assembly and manufacturing techniques, sequences, and tolerances. Complex tasks are decomposed into robot programming language commands and procedure calls using the known rules and sequences. Peeking from the bottom up, figure 1 might be considered to represent an evolution of a goal-directed robot system. The present effort is concerned with a robot programming language which fits within this system model in terms of data communications and interfaces.

A summary of the language is given in Appendices I and II. Appendix III shows several simple programming examples.

TRICCS Application and Workstation

TRICCS includes a robot programming language aimed primarily toward learning to deal with physical environment or domain knowledge base information as a next step toward the goal-directed robot system of figure 1. Its data types are physical entities such as points, directions, lines, planes, and objects, with which a robot is expected to deal. Basic operators to manipulate these data types are provided in addition to the robot system commands. The TRICCS language contains TASKs, a procedure-like feature with parameters, which allow fundamental functional units to be defined and then called to create more and more complex operations. TASKs make interactive programming (TRICCS has an interactive or immediate mode) much easier and also make TRICCS programs much easier to read. TRICCS also has provisions to control multiple arms or devices (addressing them by name) and for specifying that more than one command, possibly involving different arms, be carried out simultaneously or concurrently. It should be noted that this in itself does not provide inter-arm coordination.

A real stumbling block to the interaction of robot programming languages with domain knowledge bases has apparently been getting the physical domain knowledge into the knowledge base in a suitable form. In figure 1 this is done by a vision system which scans the scene until an object is recognized; determines what it is from a table of information on objects it is expected to encounter, and then enters the pertinent identification, location, orientation, and size data into the knowledge base. While such a 3-D vision system identifier is not currently available, it is technically feasible using either stereo cameras or laser scan techniques. Since it is desirable that an operator be capable of describing objects which are not included in the information tables, TRICCS incorporates an interactive, manual, TV-assisted domain definition scheme. This is intended only as a back-up capability for an operational system, but here it is used as a means of beginning the study of a robot programming language which deals with the physical domain as described in a knowledge base. When a 3-D vision system identifier becomes available, commands must be added to the TRICCS language to perform such functions as searching for specific objects and identifying unknown objects.

Figure 2 shows a somewhat expanded block diagram of that part of the robotic system dealt with in this report. This figure shows how TRICCS fits into the robot command and control loop and illustrates the operator input levels for full manual or teleoperator control and for command inputs as a programmer. It also indicates those parts of the system which must operate in real time to implement sensor feedback. It should be noted that, without continuous manual feedback (teleoperator mode), a set of inverse dynamics equations must be solved in real time to produce accurate arm trajectories and positioning. TRICCS also imposes significant computational loads associated with the locating and display of environmental data.

Figure 3 shows a typical remotely controlled spacecraft with several robot arms and pivoting TV cameras. This spacecraft would fly to a service or assembly station in the vicinity of a shuttle or space station. It would attach itself to the structure to be serviced or assembled and the manipulator arms used to perform the required tasks. Figure 4 shows the layout of a possible operator workstation to

be used in commanding and controlling the robotic system of figure 3 using either manual, teleoperator control or command input to TRICCS. At least two of the camera outputs are displayed on TV monitors on the console. The cameras will be controlled by a pressure-sensitive touch tablet and stylus or by a mouse. The stylus or mouse will have to contain a switch for zoom control. The display monitors must be capable of superimposing lines and alphanumeric characters over the TV picture. Use of the TV monitors for defining and displaying TRICCS program variables is described in the Interactive Definition Section. The workstation also includes a program entry keyboard and an associated alphanumeric CRT which can display up to 24 text lines. The keyboard will contain special function keys for frequently used TRICCS commands and for CRT control functions such as screen scroll. The functions of the work station will be more fully described in subsequent sections.

TRICCS LANGUAGE DATA TYPES

TRICCS is a language where all data must be defined as being of a particular type and the data types represent physical or spatial entities in the domain environment of the robot. The following is a list and brief description of TRICCS data types. Subsequent sections will describe the definition or declaration and the manipulation of these data types:

- POINT- A discrete point in space, respresented by three (x, y, z) spatial coordinates.
- LINE- The straight, directed locus between two points, represented by its end points. Note that a line has finite length and specific direction, i.e. a vector.
- DIRECTION- A spatial orientation in a particular direction, and having no length or spatial position, represented by 3 direction cosines.
- LENGTH- A scalar, real number representing a distance but having no direction.
- PATH- A series of connected lines in space, represented by a series points. PATHs may be used as a sort of manual obstacle avoidance technique.
- PLANE- A finite, bounded flat surface in space, represented by one point and one line, two lines, three points, etc.
- ANGLE- The included angle, represented by a real number, between two lines, two directions, or two planes.
- OBJECT- In its simplest form, a rectangular box in space represented by a point which locates one corner, three lengths which are the dimensions of the box, and three directions of the sides. The object, therefore, has a location and orientation and it is intended to be moved around and reoriented. Objects are made up of combinations of basic shapes, which include rods, boxes, balls, and cones. They will also have features such as grasp points and holes. The basic shape's data will include dimensions, orientation, attachment point, surface characteristics,

etc. Hole and grasp-point data will include location, orientation, and size. The description of objects for a vision system identifier is quite complex (ref. 11). A simplified description containing only location and orientation information for compound objects, etc., follows.

Data Structure for Object

```

NAME: ALFA
LOCATION: x, y, z - in world coordinates
ORIENTATION:  $\alpha, \beta, \gamma$  - in world coordinates
ELEMENT_LST: ^ELEMENT
ATTACH_LST: ^OBJECT

ELEMENT {
  TYPE: BOX, ROD, CONE, BALL
  SIZE: (a, b, c), (d, l), (d1, d2, l), d
  LOCATION: x, y, z - with respect to previous
             element or object
  ORIENTATION:  $\alpha, \beta, \gamma$  - with respect to previous
                element or object
  GRASP_PT_LST: ^GRASP_PT
  HOLE_LST: ^HOLE
  NEXT_ELEMENT: ^ELEMENT

  HOLE {
    IDENT: ALFA
    LOCATION: x, y, z - with respect to element
               coordinates
    ORIENTATION:  $\alpha, \beta, \gamma$  - with respect to element
                  coordinates
    DIAMETER: LENGTH
    NEXT_HOLE: ^HOLE

    GRASP_PT {
      IDENT: ALFA
      LOCATION: x, y, z - with respect to element
                  coordinates
      APPROACH_DIR:  $\alpha, \beta, \gamma$  - with respect to element
                     coordinates
      NEXT_PT: ^GRASP_PT
    }
  }
}

```

The fact that there are multiple grasp points and holes means that the one desired will have to be identified by name (IDENT) to a routine that picks up an object or inserts a part into a hole. The INRANGE runtime function is used to determine if the combination of location and orientation for a hole or grasp point is within the physical capability of the arm. A NIL list will denote that there are no grasp points or holes on an element of an object. If an object may be grasped anywhere, this will be designated by assigning the name "ALL" to the identifier of its grasp point. The approach direction will then be calculated from the geometry of the object. If two objects are attached to one another during assembly, a run-time library routine, ATTACH, is used to denote this via the ATTACH_LST list. This allows many objects to be bound together.

SPEED- A scalar number representing relative speed of arm or camera motion, could be expressed as levels (1..5) or as percent maximum.

SENSOR_LEV- A scalar number representing a level or sensor reading for a manipulator or end effector could be expressed as levels or as percent maximum.

BOOLEAN- logical true or false.

STRING- string of characters for prompts and messages.

TRICCS allows one-dimensional arrays of all data types -

ARRAY [0..5] of <type>.

The declaration and definition (initialization) of these data types are described in the next section.

Declaration and Definition of Variables

Variables of geometric data types may be declared and defined either statically or interactively. Static definition is accomplished from the keyboard by simply typing in values for the variable's location, orientation, etc. Interactive definition uses the CRT displays and tablet or mouse to identify locations of critical points in order to describe the variable. Fixed domain features such as tool boxes, parts bins, and assembly points may be entered directly from data files.

Static Definition

DEFINE X : POINT = 6.5, 10.2, -3.1

DEFINE Y : LINE = 6.5, 8.4, 1.3; 3.1, 5.2, 2.3


DEFINE L : LINE = Y.END; X

where Y.END is a run time routine which returns the second, or terminating point of LINE Y which equals (3.1, 5.2, 2.3).

If the definition is incomplete, the operator will be prompted for the remainder.

Interactive Definition

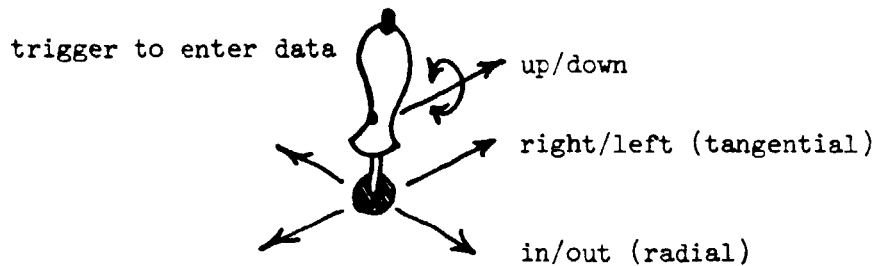
DEFINE X : POINT

Typing a "return" without the "=" and the variable's value puts the system in interactive DEFINITION mode and transfers control to the CRT displays and the tablet or mouse controllers. Small cross hairs  will be displayed on the CRT

screen superimposed over the TV picture. Variable definition is accomplished by moving this "spot in space" with a three axis hand controller. The spot is rate

**ORIGINAL PAGE IS
OF POOR QUALITY**

controlled in cylindrical coordinates (centered on the "world" coordinate system) as follows:



The spot would be displayed on the two TV monitors in perspective and could be used to locate a point in space which is "marked" with the trigger button to enter the location into the variable. The display will remain in definition mode displaying the data already entered, with lines to define the variable so far and a blinking line to show the line presently being defined. For example, to define a LINE variable, the cross hairs of both CRT screens are centered on the desired point for the beginning of the LINE and the trigger is pressed to enter that point. For a LINE definition, the system would remain in input mode until another point, the end of the LINE, is defined. The LINE variable would be displayed until the operator either verified its correctness, or moved one or both ends until it is correct. Other data types have different definition sequences required to completely define the variable. The operator will be prompted at each step as to what is required next.

The current position of the robot arm may be used to define a POINT variable and/or the current orientation of the manipulator may be used to define a DIRECTION variable:

MARK X : POINT

DIRECTION

A previously defined variable may be opened for change or redefinition by the command

CHECK X

The system will go into input mode with X displayed on the CRT screens and the locations corrected or verified as described above. To remove a variable from the program's directory:

FORGET X

This will prompt the operator with "ARE YOU SURE YOU WANT TO FORGET LINE X?" if X is of type LINE and displays X on the CRT screens as a safety precaution to the inadvertent removal of variables. Geometric variables may be displayed on the CRT screens by

SHOW X

where X may be a POINT, LINE, PLANE, OBJECT, PATH or DIRECTION. Up to three variables may be shown simultaneously, each appearing in a primary color; red, blue, or yellow with its name in the same color at the bottom of the screen. A variable may be removed from the display by

ERASE X

A listing of all current program variables and their types may be obtained by

LIST VARS

which appears on the alphanumeric CRT.

System Sensor/Actuator Names

The robotic system may consist of one or more robot arm with associated sensors and two or more remotely controlled TV cameras as shown in figure 3. The TV camera outputs are displayed on two CRT screens as shown in figure 4. Each of these controlled hardware elements must have a name or program identifier associated with them in order that the TRICCS program may direct commands to the proper device.

There are four basic data types in TRICCS which relate to external devices: ARM, SENSOR, CAMERA, and SCREEN. Together, they comprise a compound data type (record) called SYSTEM. There are also several arrays of input/output ports for device command, feedback, and video signals. These are PORTIN, PORTOUT, VIDEOIN, and VIDEOOUT. The external device variables are declared and linked to the appropriate input/output as follows:

```
DEFINE ARM LEFT = KOUT1*PORTOUT[1]
DEFINE ARM RIGHT = KOUT2*PORTOUT[2]
DEFINE SENSOR LEFT/FORCE = KIN1*PORTIN[1]
DEFINE SENSOR RIGHT/TACTLE = KIN2*PORTIN[2]
DEFINE CAMERA CAM1 = KOUT3*PORTOUT[3], VIDEOIN[1]
DEFINE SCREEN STARBRD = VIDEOOUT[1]
```

where KOUT1, KIN1, etc., are gains on the I/O signals. Actually, this may be any expression, such as

```
KOUT1*PORTOUT[1] + BIASOUT
```

Note that sensors must be associated with a previously declared arm. The definition of external devices will be performed at system set-up time and remain fixed. The statement

SHOW SYSTEM

will display the current system configuration on the alphanumeric CRT. Motion commands directed to a particular arm will be

LEFT/ <ARM COMMAND> i.e. LEFT/GO'X X

Further, there will be a "running default" in the case of multiple arms. Once the operator or program has said "LEFT/", all subsequent unprefixed arm commands will be directed to the LEFT arm until, the occurrence of a "RIGHT/" command, whereupon the default will become the RIGHT arm. Single arm systems require no prefix, regardless of the name of the arm.

Sensors which must have names include those which are fed back to the TRICCS program. These include force, torque, proximity, and tactile (touch) sensors. Since these sensors are associated with the robot arms, the conditional and loop statements utilizing their feedback (Sensor Feedback Section) must be prefixed by the appropriate arm designator or use the default. The statement "IF LEFT/FORCE <.5 THEN ..." represents an implicit read of that sensor (ref. 12). Camera/display commands are complicated by the fact that only combinations of camera and display screen may be commanded; i.e., no camera may be moved unless it is assigned to a display. The CRT screens may be named PORT, to the left, and STARBRD, to the right. Cameras are assigned to the screens with the DISPLAY command described in the Camera/Display Commands Section and command format is "PORT/" as in arm commands. Default rules are also identical to the robot arm situation. TRICCS will check to insure that only valid arm names precede arm motion commands and camera/display names precede camera commands.

Arm Motion Commands

The robot arm motions are divided into three mutually exclusive types: translation, orientation, and manipulator commands. Manipulator here denotes the hand or end effector located at the tip of the arm and used to grasp or manipulate the work piece. Translation commands involve translation motion of the tip of the manipulator in Cartesian world coordinates fixed to the robotic system environment and do not reorient the manipulator. Orientation commands change the direction in which the manipulator is pointing without moving the tip of the manipulator. These involve arm motions to compensate for the movement of the tip of the manipulator. Manipulator commands involve only the roll and the opening and closing of the manipulator itself.

Translation Commands:

GOTO X (POINT) - move the tip of the arm's manipulator to the point represented by variable X

GO Y (LENGTH) - move for the distance Y in the direction that the hand is pointing

FOLLOW Z (PATH or LINE) - go to beginning of path Z and follow all segments to its end

RETRACE Z (PATH or LINE) - go to the end of Z and follow all segments in reverse order to its beginning

Orientation Commands:

TURNTO X (DIRECTION) - reorient the hand in the direction specified by X

POINTTO Y (POINT) - reorient the hand so that it points toward the point Y

TRACK X (POINT) - continuously track a point during all subsequent translation motions

TRACKOFF or another orientation command will cancel the TRACK command.

Manipulator Commands:

ROTATE X (ANGLE) - rotate the manipulator hand about its center line axis to the angle defined by X

ROTATE Y (ANGLE) - rotate the manipulator hand through an angle of Y degrees

OPEN Z (% max or span distances) - open the manipulator fingers an additional Z distance

CLOSE Z (% MAX or span distance) - close the manipulator fingers an additional Z distance

OPENTO Z (% max or span distance) - move the manipulator fingers (open or close) to the opening span indicated by Z

Arm motion commands generate commands to the trajectory planner of figure 2. Translation commands might produce a vector in space to represent the desired position change and the orientation commands could generate a direction vector about which the manipulator hand is to rotate with its magnitude representing the total angle to rotate through.

Camera/Display Commands

There are a minimum of two and no maximum number of TV cameras associated with the teleoperator/robot system. These are used in pairs (displayed on two TV screens) to locate and identify objects for the teleoperator system and TRICCS program. The cameras may be moved manually by hand controllers, voice commands, or even eye movements. The cameras may be mounted on arms of their own or on the working arms to enable better angles of vision. The cameras are assigned to TV screens and may be moved under program control. The two CRT screens may be named port and starboard, PORT and STARBRD, for the left and right screen, respectively. The cameras will also have names, say CAM1-CAM6, for example. To display the output of CAM1 on the left screen:

DISPLAY CAM1 ON PORT

The camera motion commands then refer to the camera displayed on the screen addressed

PORT/LOOKAT K (POINT) - points the camera displayed on the PORT screen (CAM 1) toward the point X

LOOKAT LEFT - points CAM1 to tip of LEFT arm's manipulator

LOOKIN Y (DIRECTION) - points CAM1 in the direction Y

Screen-oriented pitch and yaw camera commands (which correspond to the manual commands using a camera control tablet or mouse) are

PORT/TILT A (ANGLE) - move CAM1 up or down through the angle A

PAN B (ANGLE) - move CAM1 left or right through the angle B

and

ZOOM (<percent max>) - zoom CAM1 in through the specified percentage of zoom range

ZOOMTO (<percent max>) - zoom CAM1 in or out to the specified percent of maximum zoom setting

The camera motion commands entered from the keyboard override any manual inputs except during data definition but the manual mode is always active. The camera name associated with each TV display will be shown at the top of each screen highlighted in reverse video.

Tasks

TRICCS TASKs are simply a defined group of commands which perform a specific function, similar to subroutines and procedures. TASKs do not have local data like procedures, but may be invoked with parameters or arguments. Up to 8 versions of a TASK may exist specified by an optional version number. The appropriate commands. Tasks may be defined by

DEFINE TSKX (PARM1:TYPE,PARM2:TYPE...) : TASK

followed by the commands and terminated by an "END". Tasks are compiled and stored for later invocation or modification by the command SAVE.

The command

SHOW TSKX (version number)

displays the text of TASK TSKX on the alphanumeric CRT screen associated with the keyboard, whereupon it may be modified (edited). The optional version number indicates which version is desired. After editing, a task may be compiled and both the source and compiled code saved by

SAVE TSKX (<version number>)

The command

FORGET TSKX (<version number>)

removes both the source and compiled code for the indicated version of TSKX library.

Tasks are called by

TSKX (PARM1, PARM2)

which calls the highest numbered version. Other versions may be specified by

```
USE VERSION <version number> FOR TSKX
```

either in the calling program or entered from the keyboard prior to running the program.

POINT type parameters may be literals or expressions

```
TSKX ((PTX + (.5, 2.1, 5.3)), PT1)
```

Simple or routine operations such as pick up, place, operate a button or switch, etc., may be defined as TASKs and called with parameters specifying the location of the activity. Parameters may be of any data type or the name of another TASK. TASKs may be built hierarchically to define higher and higher level commands. A library of basic TASKs will be available and some mechanism for user libraries to be included will be developed.

Other Language Features and Sensor Feedback

The TRICCS language includes the normal conditional and loop features found in most computer languages:

- (1) IF <conditional> THEN
 commands
 .
 .
ELSE
 commands
 .
 .
END
 - (2) DO WHILE <conditional>
 commands
 .
 .
END
 - (3) REPEAT
 commands
 .
 .
UNTIL <conditional>
- any block may contain an
- (4) EXITIF <conditional>

where the conditional expression may involve sensor signals such as force or proximity which are fed back from the manipulator or arm. The sensor feedback is also used in the following arm and manipulator motion commands:

GO UNTIL <conditional> - move in the direction the hand is pointing
until the condition is satisfied

OPEN UNTIL <conditional> - open the manipulator fingers until the
condition is met

CLOSE UNTIL <conditional> - close the manipulator fingers until the
condition is met

ROTATE UNTIL <conditional> - rotate the manipulator until the
condition is met

For example,

LEFT/CLOSE UNTIL FORCE>=GLASSBRK-.3

RIGHT/GO UNTIL TOUCH

OPEN UNTIL NOT TOUCH

ROTATE UNTIL TORQUE>TWISTOFF

General Commands

Any motion commands may be interrupted with

PAUSE

.
.
.

RESUME

PAUSE puts the system into IMMEDIATE mode described in the next section, System Operation, where interactive commands may be executed. Typing a command during a PAUSED condition will result in a system prompt asking whether you plan to continue the previous command. A "yes" will save the rest of that command or program for execution after completion of the command which has been typed (and a RESUME). This effectively permits a user to insert steps while keeping his main command sequence in "hold."

PAUSE Δt is a programmed pause of Δt seconds, where
 Δt is a real number.

Speed specification:

An optional speed attribute, in terms of percent maximum, may be imposed on any motion command by

LEFT/GOTO X AT Z (SPEED)

The specified speed then becomes default speed for that arm until another speed specification is directed to that arm. Direct speed changes may be applied interactively by

LEFT/AT Z

will modify arm speed in mid-maneuver.

Combined commands:

TRICCS commands are normally executed sequentially; one command is completed before the next is initiated. A currently executing TASK will complete before an interactively entered command can begin. One exception to this is interactively no matter what else is going on. Some commands, involving different elements of a single arm, may be designated to occur simultaneously by

GOTO X AND POINT TO Y

The parser will check for a valid combination which involves no more than one each of translation, orientation, and manipulator commands for a single arm. Semantically, the commands will be timed to complete simultaneously.

The next question involves simultaneous commands for more than one arm, the arm coordination problem. Like the obstacle avoidance problem, entered camera commands which execute immediately the multi-arm coordination implementation involves extensive logic (involving avoidance) which is considered to be beyond the scope of this study of TRICCS. It can be said, however, that such commands could easily be incorporated in TRICCS from a language or syntactic point of view as:

LEFT/GOTO X AND RIGHT/GOTO Y

System Operation

The TRICCS robotic system illustrated in figure 2 and using the workstation described in figure 4 is able to operate in several modes:

TELEOPERATOR - the lowest level of control where continuous manual control stick inputs are supplied to the robot arms and cameras

IMMEDIATE - an interactive mode where each TRICCS command is executed immediately and the system waits for another.

DEFINITION - the data definition for the domain knowledge base where the cameras are selectable from the keyboard and are pivoted with the hand controllers or touch tablets.

DEBUG - A checkout mode where arm motions are not actually executed, but are displayed using a computer-generated graphic display which may be superimposed over the TV display for additional realism.

MONITOR - A dynamic debug mode where the next command is displayed and the system pauses waiting for manual verification before executing it.

PROGRAM - The system is executing under program control and manual inputs are restricted to PAUSE and manual camera commands.

The hierarchy of operation of these modes is illustrated in figure 5. For full system flexibility, the operational modes should be able to operate in conjunction with one another. There is no reason why additional data could not be defined while a programmed activity is being performed. Also, PROGRAM mode should be capable of pausing while another mode such as IMMEDIATE is used to do extra things or MONITOR is entered to provide more checking for a few steps and then the program resumed at any point.

When a program name is invoked, PROG1(PARM1,PARM2), the system is in PROGRAM mode. This may be interrupted by a programmed or interactively entered PAUSE, which puts the system in a combination of TELEOPERATOR and IMMEDIATE mode with the program execution suspended or on "hold" as described in the previous section. If no program is executing, the system is also in the TELEOPERATOR/IMMEDIATE mode. Here, either manual arm inputs or keyboard language commands may be entered, with the keyboard commands taking precedence. The system enters DEFINITION mode upon receipt of a DEFINE command. DEFINITION mode does not suspend a program which may continue executing while variables are being defined. The DEFINITION mode may be exited and the variable definition abandoned by the command "MODEOFF". DEBUG mode works with both IMMEDIATE and PROGRAM modes. The command "DEBUG" is entered when no program is running and sets up the displays and disables the arm commands. MONITOR mode works only during PROGRAM mode and the command "MONITOR" is entered at any time during program execution to put the system into what is basically a step mode. Both DEBUG and MONITOR modes are exited by the command "MODEOFF". This command may be entered at any time during program execution to exit MONITOR mode, but must be entered when no program is executing for DEBUG mode.

For operation in a space environment with new and unexpected situations and incompletely specified tasks, a robotic system will require a significant amount of manual monitoring and intervention. This manual interaction, to be efficient and effective, must be at the highest possible level with the system providing programmable support. The TRICCS hierarchical TASK structure with preprogrammed, modifiable, parametrically callable functions to support manual and expert systems (TASK descriptions would be a part of the fact/rule knowledge base of figure 1) is consistent with this paradigm. In practice, robotic system operations in space will involve the off-line programming and checkout of both generalized and specific task functions to support on-line operations. Manual modes will provide backup capability for reliability and as the evolving robotic system becomes more sophisticated, less monitoring will be required and manual intervention will be at a higher level. Both the TRICCS language and operational modes are aimed at the support of an evolving, space operations-oriented robotic system. Its purpose is to investigate areas of system evolution such as knowledge bases, sensor feedback, and expert systems and also to develop manual interfaces required for reliability in dealing with these areas of evolution and with the new and unexpected situations imposed by the space environment.

It should be noted that TRICCS is not a "language" in the sense of requiring a compiler and full language support. In its simplest form, INTERACTIVE mode, it is a command parser/interpreter running in non-real time as shown in figure 2. It could be written in PASCAL or Ada and called by operator input to generate proper motion commands for the robot arms and TV cameras. Its run-time support routines, which will be extensive, would also be written in PASCAL or Ada. To implement "precompiled" tasks or subroutines, the source code could be converted to an intermediate P-code-like form and saved on a library file. The interpreter would then execute the P-code. However, since the TRICCS language involves direct hardware communication via sensor feedback, the execution portion may have to run in a real-time environment.

Also, the requirement for multiple arm operations and coordination almost dictates concurrent processing. It would make little sense to develop a concurrent processing capability specifically for a robot programming language. Thus, TRICCS will best be implemented as packages and tasks in Ada. The TRICCS TASKs would become Ada tasks and could thus run concurrently, providing a powerful facility for inter-arm coordination.

This would also make the TRICCS language readily expandable and extensible. The modularity afforded by Ada OBJECTS would make it easier to produce a robot configuration independent system which would also benefit from Ada's portability. References 6 and 13 investigate Ada implementations for robot systems and programming languages.

Concluding Remarks

This report describes a proposed robot programming language aimed at the investigation of domain knowledge base interaction, manual interface, and sensor feedback for space related robotic systems. These are essential for a goal-directed robot system which provides the operational flexibility required for space operations. The system is designed for operation at several levels from a completely manual teleoperator to a fully programmed system interacting with an expert system/planner. All modes of system operation, including generation of the domain knowledge base, have manual backup capabilities. Several operational modes are provided for interactive debug, checkout, and monitoring. Sensor feedback and multi-arm coordination requires real time, concurrent processing capability so that the TRICCS system should be implemented as packages and concurrent tasks in Ada.

The TRICCS concept has been developed at LaRC as a robot programming language research effort. It is not implemented either in hardware or software. It has become clear during this study that several areas basic to the realization of a system such as TRICCS require more work. These are: 1) the definition of a world model knowledge base structure consistent with vision system identifier, programming language and other interface requirements, 2) a 3-D vision/identification system capable of generating the world model and relating its information to CAD/CAM type of data. Attention to these two areas is essential before a goal-directed robotic system such as TRICCS is realizable.

REFERENCES

1. Soroka, Barry I.: "What Can't Robot Languages Do?," 13th International Symposium on Industrial Robots & Robots 7, April 17-21, 1983, Chicago, Illinois.
2. Gruver, William A.; Soroka, Barry T.; Craig, John J. and Turner, Timothy L.: "Evaluation of Commercially Available Robot Programming Languages," 13th International Symposium on Industrial Robots & Robots 7, April 17-21, 1983, Chicago, Illinois.
3. Shimano, Bruce E.; Geschke, Clifford C.; Spaulding, Charles H., III and Smith Paul G.: "A Robot Programming System Incorporating Real Time and Supervisory Control: VAL II". Robots 8 Conference Proceedings, Volume 2; June 4-7, 1984, Detroit, Michigan.
4. Bonner, Susan and Shin, Kang G.: "A Comparative Study of Robot Languages". IEEE Computer, December 1982.
5. Mujtaba, M. S.; Goldman, R. and Binford, T.: "The AL Robot Programming Language," Computers in Engineering 1982, Volume 2: Robots and Robotics, Proceedings of the Second International Computer Engineering Conference, Aug. 15-19, 1982, San Diego, California.
6. Volz, R. A.; Mudge, T. N. and Gal, D. A.: "Using ADA as a Robot System Programming Language," 13th International Symposium on Industrial Robots and Robots 7, April 17-21, 1983.
 - Kirschbrown, Richard H. and Dorf, Richard C.: "KARMA - A Knowledge-Based Robot Manipulation System: Determining Problem Characteristics." Robots 8 Conference Proceedings, Volume 2, June 4-7, 1984, Detroit, Michigan.
8. Inoue, H.; Ogasawara, T.; Shiroshita, O., and Naito, O.: "Design and Implementation of High-Level Robot Language," Proceedings of 11th International Symposium on Industrial Robots, October 7-9, 1981, Tokyo, Japan.
9. Barbera, Anthony J.; Fitzgerald, M. L.; Albus, James, S. and Haynes, Leonard S.: "RCS: The NBS Real Time Control System," Robots 8 Conference Proceedings, Volume 2, June 4-7, 1984, Detroit, Michigan.
10. Hayward, Vincent and Paul, Richard P.: "Robot Manipulator Control Using the "C" Language Under UNIX." Proceedings of IEEE Workshop on Languages for Computer Automation, November 7-9, 1983, Chicago, Illinois.
11. Kakazu, Y.; Okino, N. and Utsumi, K.: "Pattern Recognition Problem on Modeled 3-D Geometry." 11th International Symposium on Industrial Robots, October 7-9, 1981.
12. Chern, Ming-Yang; Chern, Mei-Ling; and Moher, Thomas G.: "A Language Extension for Sensor-Based Robotic Systems", IEEE Workshop on Languages for Computer Automation, November 7-9, 1983, Chicago, Illinois.
13. Buzzard, G. D.; and Mudge, T. N.: "Object-Based Computing and the Ada Programming Language." IEEE Computer, March 1985.

APPENDIX I

RUNTIME FUNCTIONS

A set of geometric operation and conversion utilities are provided in the form of standard functions:

INTERSECTION (LINE, LINE)→POINT or nil
(PLANE, PLANE)→LINE or nil

NORMAL (LINE)→PLANE
(PLANE)→DIRECTION

LENGTH (LINE)→LENGTH

RECIPROCAL (DIRECTION)→DIRECTION

DIRECTIONOF (LINE)→DIRECTION

INCLUDANG (LINE, DIRECTION)→ANGLE
PLANE, ETC

LINE (POINT, POINT)→LINE

PLANE (POINT, POINT, POINT)→PLANE
(POINT, LINE)

PATH (POINT, LINE, PT, ..ETC)→PATH

LINE.BEG (LINE)→POINT

LINE.END (LINE)→POINT

MIDPT (LINE)→POINT

RELOCATE (OBJECT) - calculates new location and orientation for an
object

ATTACH (OBJECT1, OBJECT2) - defines a physical connection between two
objects such that when one is moved (relocated)
the other is also moved

(POINT)
(LINE)→BOOLEAN - indicates whether the variable is completely
INRANGE (PATH) within range (reach) of a robot arm
(HOLE)
(GRASP PT)

GRASP Y (Force) - close manipulator until force builds up to specified level

CONTACT Y (Force) - move forward in direction hand is pointing until contact is made
with force level specified, sensed by force sensors

RELEASE - opposite of GRASP or CONTACT

CENTER - centers an object between the fingers by moving the arm in
the proper direction once one finger has made contact

APPENDIX II TRICCS SUMMARY SHEET

DATA TYPES: POINT, ANGLE, PLANE, PATH, OBJECT, LENGTH, DIRECTION, ANGLE, SPEED, FORCE

CODE BLOCK: TASK (PARAMETERS)
 FORGET <NAME>
 CHECK <NAME>
 MARK <NAME> :POINT
 :DIRECTION

DATA DISPLAY: SHOW <NAME>
 ERASE <NAME>
 LIST VARS

DEFINE ARM <NAME> = <PORTOUT EXPRESSION>

DEFINE SENSOR <ARMNAME>/<NAME> = <PORTIN EXPRESSION>

DEFINE CAMERA <NAME> = <PORTOUT EXPRESSION>, VIDEOIN<CHANNEL>

DEFINE SCREEN <NAME> = VIDEOOUT<CHANNEL>

SHOW SYSTEM

ARM MOTION:

TRANSLATION: GOTO <POINT>
 GO <LENGTH>
 FOLLOW <PATH>
 RETRACE <PATH>

ORIENTATION: TURNTO <DIRECTION>
 POINTO <POINT>
 TRACK <POINT>
 TRACKOFF

MANIPULATOR: ROTATETO<ANGLE>
 ROTATE <ANGLE>
 OPEN <% MAX>
 CLOSE <% MAX>
 OPENTO <% MAX>

CAMERA/DISPLAY ASSIGNMENT: DISPLAY <CAMERA NAME> ON <SCREEN NAME>

CAMERA MOTION: LOOKAT <POINT>
 LOOKIN <DIRECTION>
 TILT <ANGLE>
 FAN <ANGLE>
 ZOOM <% MAX>
 ZOOMTO <% MAX>

```

TASKS:  DEFINE <NAME> (PARM: TYPE: ...): TASK
        END

        <TASK> (PARMS)

        CANCEL <TASK>

        SHOW <TASK>

CONDITIONAL:  IF <condition> THEN..ELSE..END

            DO WHILE <condition>..END

            REPEAT..UNTIL <condition>

            EXITIF <condition>

SENSOR FEEDBACK:  GO UNTIL <condition>
  (manipulator)
            OPEN UNTIL <condition>

            CLOSE UNTIL <condition>

            ROTATE UNTIL <condition>

SPEED:  AT <SPEED>

COMBINED:  <TRANSLATION> AND <ORIENTATION> AND <MANIPULATOR>

GENERAL:  PAUSE

            RESUME

MODE CONTROL:  DEBUG

            MONITOR

            MODEOFF

RUNTIME FUNCTIONS:  INTERSECTION (<LINE>, <LINE>)+ <POINT>
                    (<PLANE>, <PLANE>)+<LINE>

                    NORMAL (<LINE>)+<PLANE>
                        (<PLANE>)+<DIRECTION>

                    LENGTH (<LINE>)+<LENGTH>

                    RECIPROCAL (<DIRECTION>)+<DIRECTION>

                    DIRECTIONOF (<LINE>)+<DIRECTION>

                    INCLUDANG (<LINE>, <PLANE>)+<ANGLE>
                        (<DIRECTION>, ETC)

                    LINE (<POINT>, <POINT>)+<LINE>

```

```

PLANE (<POINT>, <POINT>, <POINT>)+<PLANE>
      <POINT>, <LINE>

PATH (<POINT>, <LINE>, <POINTS>, ..)+ <PATH>

LINE. BEG (<LINE>)+<POINT>

LINE. END (<LINE>)+<POINT>

MIDPT (<LINE>)+<POINT>

RELOCATE (<OBJECTS>)

INRANGE <POINT>
        <LINE>+ <BOOLEAN>
        <PATH>

GRASP (<FORCE>)

CONTACT (<FORCE>)

RELEASE

CENTER

```

APPENDIX III

TRICCS PROGRAM EXAMPLES

Included here are two simple examples of the type of robot arm tasks which may be programmed in TRICCS. They are intended to illustrate the flavor of TRICCS in terms of language capability, run-time functions, and hierarchical program structure. More complex tasks will obviously require more extensive run-time support in terms of modular, preprogrammed functions, but these fit well within the framework of TRICCS.

A. Push a Panel Button

BUTTON_A - is a POINT representing the button to be operated

PANEL - is a PLANE representing the panel in which the button is mounted

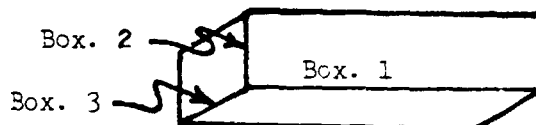
```
DEFINE PUSHBUTTON (BUTTON_A: POINT: PANEL: PLANE) : TASK
```

```
    GOTO BUTTON_A
    POINTIN (NORMAL (PANEL))
    OPEN 0
    CONTACT BUTTON_F
END
```

where BUTTON_F is the force level required to operate the button

B. Pick Up and Move a Block

A rectangular object may be represented by



three lines as shown. These will be referred to as Box. 1, Box. 2, and Box. 3. It should be noted that the coordinates of these lines change when the object is moved, as defined by the function RELOCATE.

```
DEFINE MOVE (MYBOX: OBJECT; NEWLOC: POINT; NEWOR: DIRECTION): TASK
    GET (MYBOX)
    PUT (MYBOX, NEWLOC, NEWOR)
END
```



```

DEFINE GET (MYBOX: OBJECT): TASK
  GOTO MIDPT (MYBOX.1)
  POINTIN DIRECTION (MYBOX.2)
  ROTATE TO DIRECTION (MYBOX.2)
  OPEN TO LENGTH (MYBOX.3) - FUDGE
  GO MIN(LENGTH(MYBOX.2) - FUDGE, MAXGRP)
  GRASP BOX_F
END

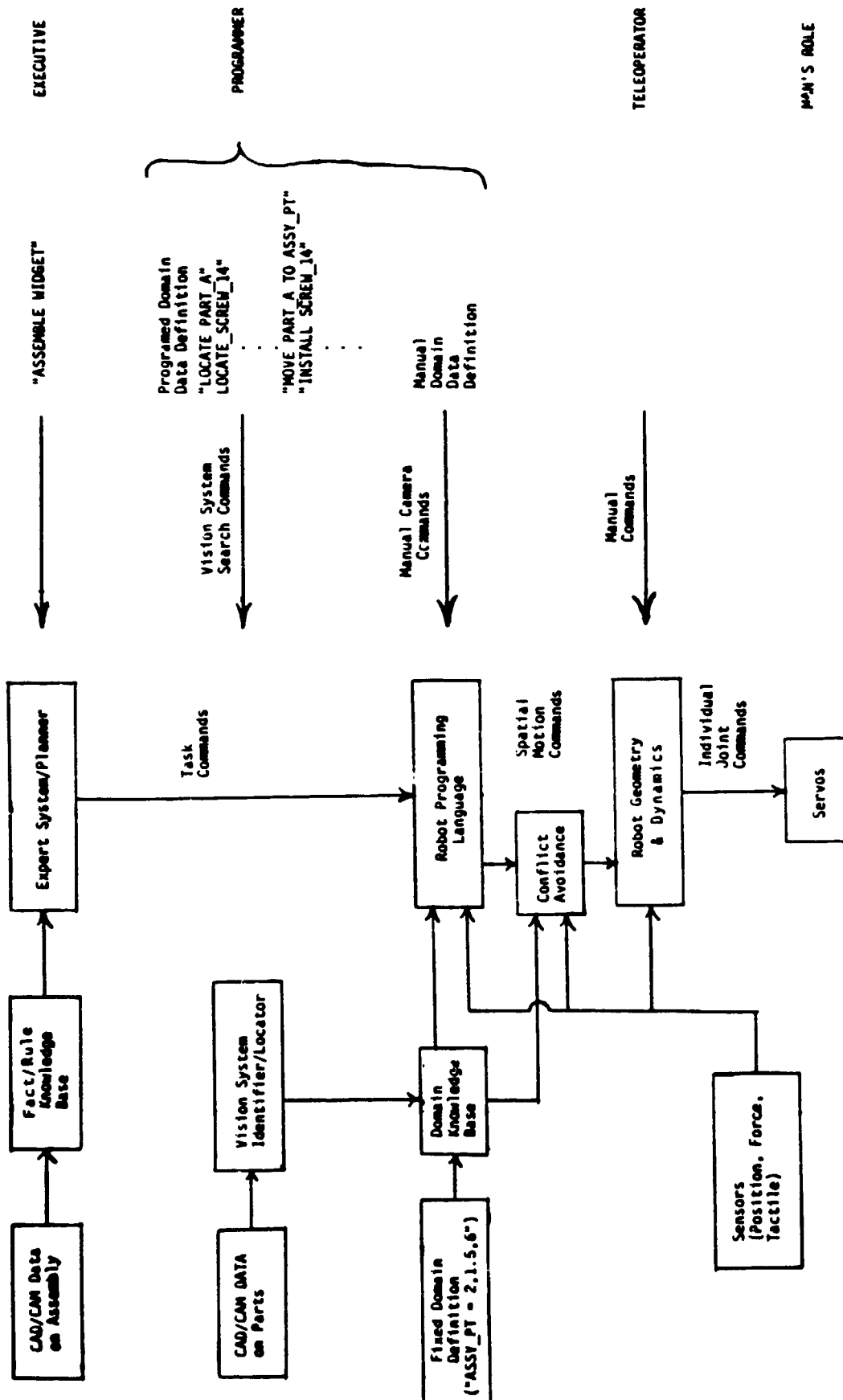
```

where FUDGE is the desired manipulator clearance
and BOX_F is the force required to handle the box

```

DEFINE PUT (MYBOX: OBJECT; NEWLOC: POINT; NEWOR: DIRECTION): TASK
  GOTO NEWLOC
  POINTIN NEWOR
  RELOCATE (MYBOX)
  RELEASE
  GO (-LENGTH (MYBOX.2))
END

```



ORIGINAL PAGE IS
OF POOR QUALITY

Figure 1.- Goal-Directed Teleoperator/Robot System

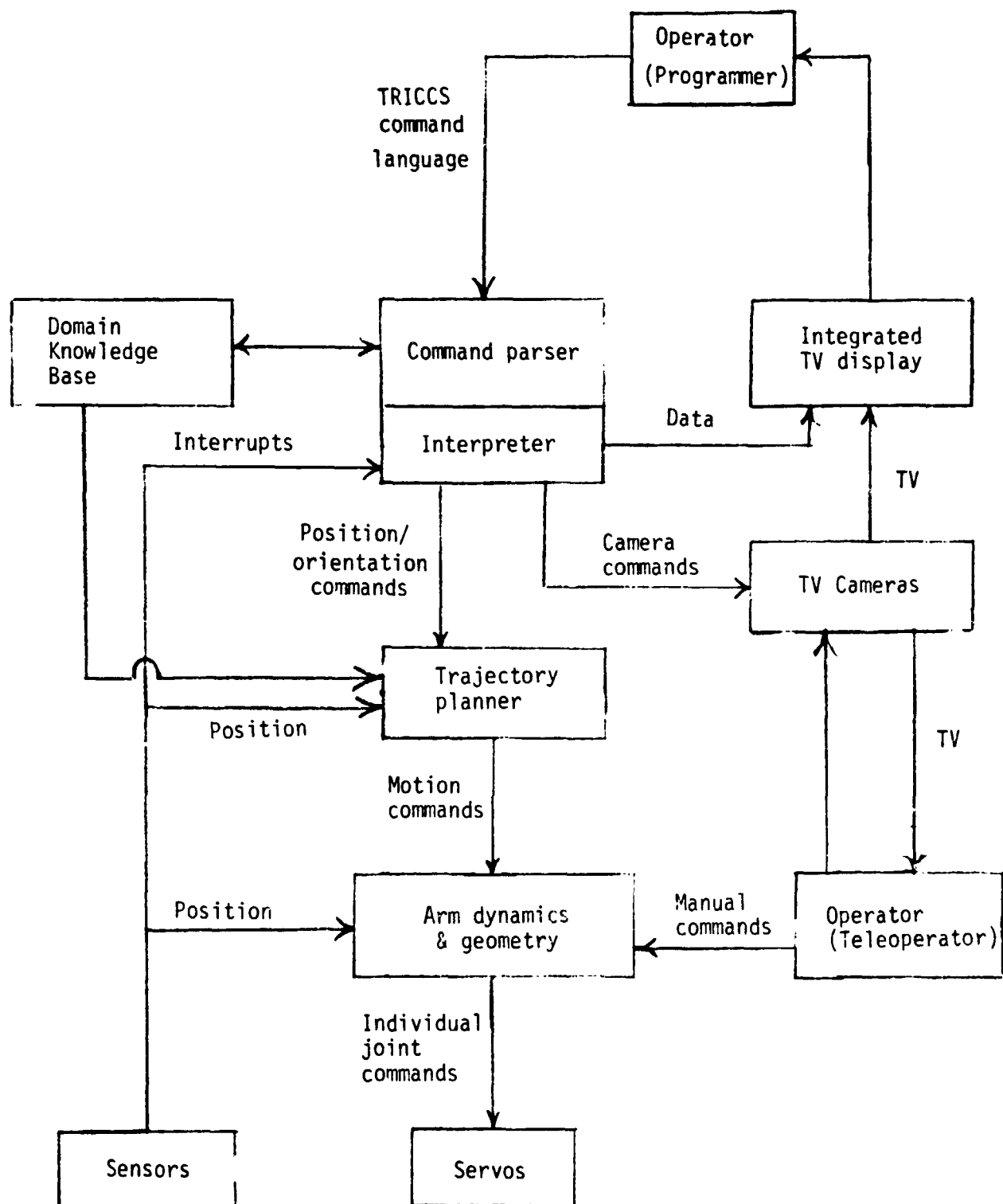


Figure 2.- Teleoperator/Robot Integrated Command and Control System (TRICCS)

ORIGINAL PAGE IS
OF POOR QUALITY

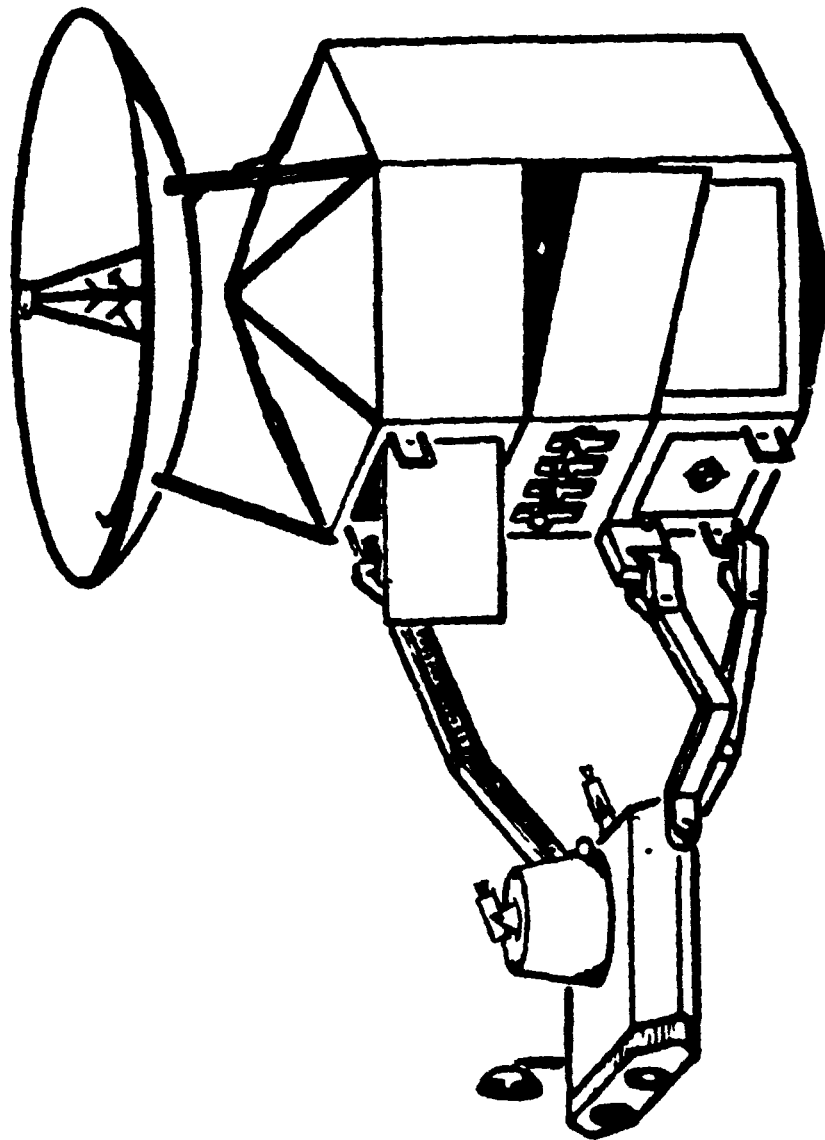


Figure 3.- Typical Space-Related Test and
Repair Robot

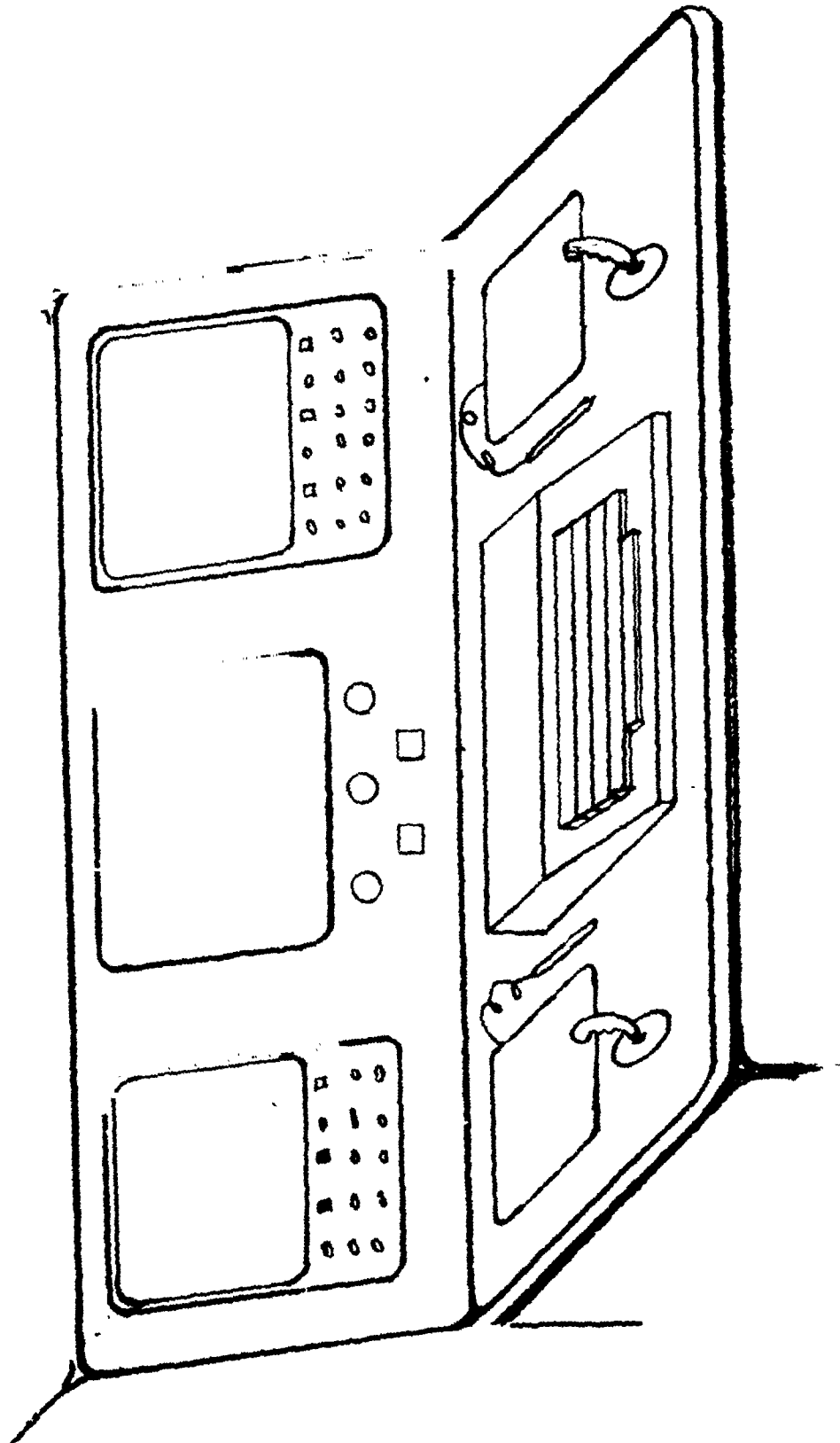


Figure 4.- TRICCS Workstation Layout

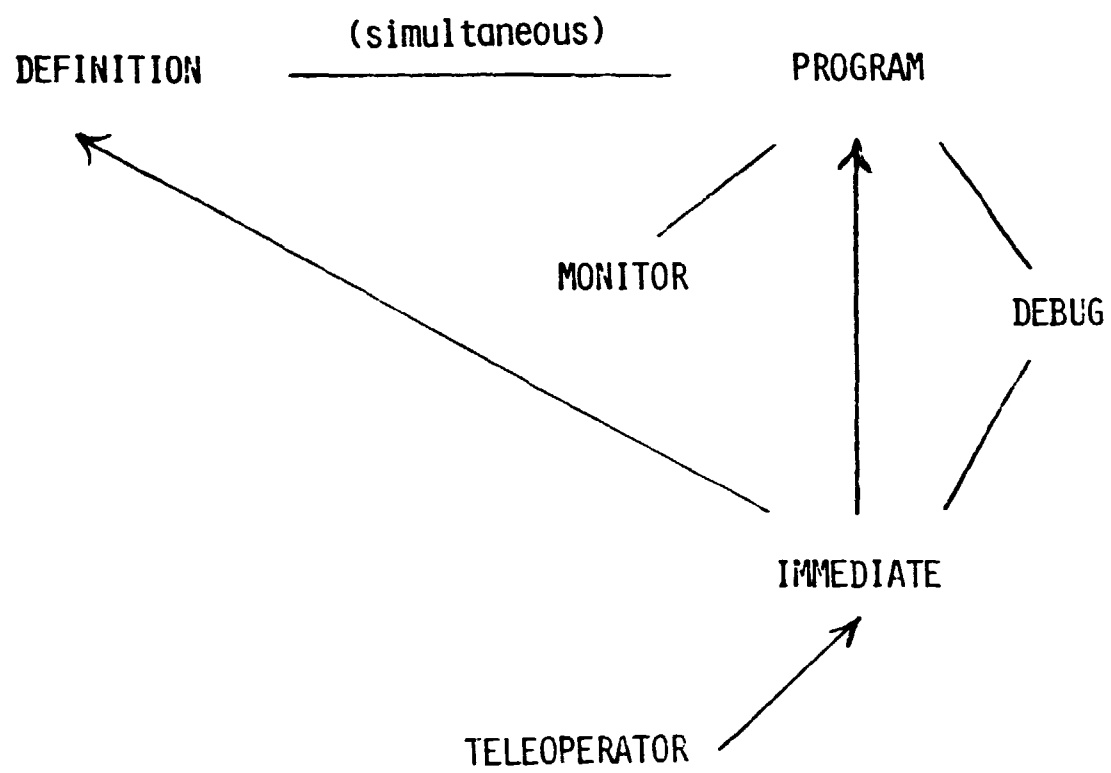


Figure 5.- Hierarchy of TRICCS Operational Modes

1 Report No. NASA TM-87577		2 Government Accession No		3 Recipient's Catalog No	
4 Title and Subtitle TRICCS: A Proposed Teleoperator/Robot Integrated Command and Control System for Space Applications				5 Report Date July 1985	
				6 Performing Organization Report No 506-54-63-01	
7 Author(s) Ralph W. Will				8 Performing Organization Report No	
9 Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665				10 Work Unit No	
				11 Contract or Grant No	
12 Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546				13 Type of Report and Period Covered Technical Memorandum	
				14 Army Project No.	
15 Supplementary Notes					
16 Abstract <p>Robotic systems will play an increasingly important role in space operations. This paper describes an integrated command and control system based on the requirements of space-related applications and incorporating features necessary for the evolution of advanced goal-directed robotic systems. These features include: interaction with a world model or domain knowledge base, sensor feedback, multiple-arm capability and concurrent operations. The system makes maximum use of manual interaction at all levels for debug, monitoring, and operational reliability. It is shown that the robotic command and control system may most advantageously be implemented as packages and tasks in Ada.</p> <p style="text-align: right;">ORIGINAL OF POOR QUALITY</p>					
17 Key Words (Suggested by Author(s)) Robotics, Robot Programming Languages, Programming Languages, Automation				18 Distribution Statement Unclassified - Unlimited Subject Category 61	
19 Security Classif (of this report) Unclassified	20 Security Classif (of this page) Unclassified	21 No of Pages 30	22 Price A03		